НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Формування інформаційних портретів морських об'єктів на підставі аналізу змісту відповідних Веб-сайтів»

Виконав:
студент IV курсу, групи ТМ-62
Черкас Богдан Тарасович _____

Керівник:
Асистент Швайко Валерій Григорович _____

Рецензент: _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп'ютерні науки та інформаційні технології

Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
"___"_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Черкас Богдан Тарасович
(прізвище, ім'я, по батькові)

1. Тема роботи «Формування інформаційних портретів морських об'єктів на підставі аналізу змісту відповідних Веб-сайтів»

керівник роботи Швайко Валерій Григорович
(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від "25"052020р. № 1268-с

2. Строк подання студентом роботи 15.06.2020

3. Вихідні дані до роботи платформа Intellij Idea, мова програмування Java з бібліотеками Jsoup, StringUtils, Swing, Apache POI та StringUtils.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Розробити алгоритм для моделювання гідроакустичного сигналу. Реалізувати код системи для цього алгоритму. Створити зручний користувацький інтерфейс для вводу початкових данних. Витягнути та зберігти дані про кораблі в файл типу Excel.

5. Перелік ілюстративного матеріалу

«Кінцева схема роботи», «Приклад програми на Java», «Приклад використання бібліотеки Jsoup», «Приклад файлу pom.xml», «Приклад використання бібліотеки StringUtils».

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”14” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	14.10.2019	
2.	Вивчення та аналіз задачі	14.10.2019-23.12.2019	
3.	Розробка архітектури та загальної структури системи	02.02.2020-03.03.2020	
4.	Розробка структур окремих підсистем	04.03.2020-14.04.2020	
5.	Програмна реалізація системи	15.04.2020-19.05.2020	
6.	Оформлення пояснювальної записки	20.05.2020-03.06.2020	
7.	Захист програмного продукту	30.05.2020	
8.	Передзахист	7.06.2020	
9.	Захист	18.06.2020	

Студент _____ Богдан ЧЕРКАС
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Валерій ШВАЙКО
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою цієї роботи є створення зручного у використанні парсеру(додатку, що виіягує та зберігає данні).

Парсер має витягати данні про кораблі, структурувати та зберігати їх в excel файл. Додатково розроблено надбудову - інтерфейс, що спрощує взаємодію зі створеним додатком.

Загальний обсяг роботи: 54 сторінки, 22 ілюстрації.

Ключові слова: парсер, excel, надбудова, додаток.

ABSTRACT

The purpose of this work is to create a parser(the application, which extract and store data).

Parser should extract the data about ships, structure them and save. Additionally, an add-on has been developed that simplifies the interaction with the created application. Total volume of work: 54 pages, 22 illustrations.

Key words: parser, excel, add-in, application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

ПАРСЕР – програма, перетворюючої вхідні дані (як правило, текст) в структурований формат

UI – User Interface

ВСТУП

Витягання даних та їх структуроване збереження дає можливість отримати базу даних в тій чи іншій галузі, для чого це б не було потрібно.

Важливим етапом існування тієї чи іншої справи є змога отримувати та досліджувати інформацію автоматизовано. Наприклад, правильне та розумне конкурування у бізнесі, впровадження чогось нового та кращого, більш вигідного для того чи іншого споживача.

Ціллю витягання та структурування даних можуть бути висновки, щодо конкурентів, що дає можливість покращити свій бізнес, досягнути високої ефективності. Іноді це дуже допомагає у створенні стратегії розвитку своєї справи, залучення нових клієнтів та/або повернення старих. Не дарма кажуть, що інформація володіє світом.

Процесний аспект полягає у тому, що спеціально навчені люди використовують парсери та досліджують інформацію на основі якої роблять висновки та подальші дії, встановлюють цілі та задачі, забезпечують досягнення цих задач за допомогою планування.

Функціональний аспект передбачає виконання наступних функцій у процесі управління системою парсингу: формування списку сайтів та ключових слів або назв продуктів, правильна структуризація, сортування, фільтрація, облік, аналіз даних, навчання. За допомогою виконання цих функцій персонал маркетингового відділу забезпечує умови та організовує ефективне конкурування з іншими бізнесами, що займається тим самим що і ви для досягення високої ефективності, ґрунтуючись на інформаційних ресурсах, представляє найбільш ефективні припущення для реалізації встановлених цілей.

Наочність та автоматизація спрощує всі процеси, у тому числі і маркетинг . Системи для парсингу включають у себе функції і можливості людини, пришвидшені в десятки сотень, а то й тисяч разів для витягання, структурування та аналізу даних.

Швидкий розвиток обчислювальних можливостей сучасних ЕОМ та можливостей програмування сприяв стрімкому розвитку парсингу. Створення таких додатків було зумовлене великою кількістю інформації, накопиченої роками, яку людина не в змозі опрацювати самотушки.

Для прикладу, на сайтах провідних брендів одягу кількість продуктіва може сягати десятків, а то й сотень тисяч товарів. Людина просто не в змозі опрацювати це, без допомоги машини.

Подібне програмне забезпечення може бути корисним, як звичайним аматорам, так і будь-якому бізнесу та навіть владі та таким структурам як поліція і їм подібним.

1 ЗАДАЧА СТВОРЕННЯ ПАРСЕРУ

За багато років розвитку люди зрозуміли, що мозок набагато повільніший та не може так швидко, як парсер, витягувати, структурувати, та збегігати дані з будь-яких джерел. Тому і було прийнято рішення автоматизувати синтаксичний аналіз інформації, а для зручності користування - створити певний текстовий документ з по кроковими інструкціями або зрозумілий, навіть не досвідченому користувачеві інтерфейс.

У даному розділі розкрита сутність дипломного завдання та обумовлені задачі, що мали бути виконані у процесі виконання. Також описана мета, проблеми її досягнення,вимоги до вихідного програмного продукту.

1.1 Мета створення парсеру

Метою цієї дипломної роботи є витягання інформації про кораблі з певного веб-сайту, її структурування та збереження для будь-яких подальших цілей та зручного перегляду.

Дана задача обумовлює такі етапи роботи:

— пошук відповідного веб-сайту;

— дослідження веб-сайту;

— розробка самого парсеру;

— розробка інтуїтивно зрозумілого інтерфейсу користувача.

Програмний продукт передбачає фунціонал для пошуку по ключовому слову на веб-сайті. Завдяки цьому буде знайдено всі відповідні позиції(у нашому випадку - кораблі) та витягнуті всі посилання на них, знайдені після пошуку. А вже після цього, після відтворення запиту на кожне посилання буде витягнута

інформація про кожну позицію. Програмний продукт повинен:

— виконувати пошук по ключовому слову ідентично до того, як це робиться на веб-сайті;

— витягати посилання тільки на позиції, знайдені після пошуку;

— робити запити на кожну позицію;

— витягати всю необхідну інформацію, структурувати та зберігати її.

1.2 Вхідні дані

Розроблений програмний продукт має отримати інформацію про позиції, що будуть знайдені після пошуку по ключовому слову. Відповідно, вхідними даними буде ключове слово, за яким буде зроблено пошук та шлях, за яким буде збережено файл с інформацією.

Від користувача не вимагається ніяких додаткових вхідних даних. Достатньо лише відкрити створену програму, ввести ключове слово у поле та натиснути “start”. Система сама сповістить про закінчення роботи. Це робить роботу з системою доволі простою та не потребує спеціальних навичок.

Завдяки невеликій кількості вхідної інформації розроблюваний додаток не має викликати дискомфорту у користувача під час його використання.

1.3 Компоненти системи

У даній розробці дипломної роботи можна виділити головні частини, що забезпечують працездатність системи:

— парсер;

— інтерфейс до парсеру.

Розроблений додаток надає користувачеві можливість отримати важливу інформацію з веб-сайту у лічені хвилини для подальшого її дослідження.

1.4 Потенційні користувачі

Система, розроблена у ході виконання цієї бакалаврської дипломної роботи може бути корисна для людей, яким потрібна інформація щодо тих чи інших кораблів. Система не вимагає у користувача спеціальних навичок володіння ПК, тому може бути використана людиною, що не користується комп’ютерами у повсякденному житті.

2 ЗАСОБИ РОЗРОБКИ

Робота по завданню поділяється на дві частини:

— розробка парсеру;

— розробка інтерфейсу.

Робота з розробки парсеру проводилась з використанням інтегрованого середовища розробки Intellij IDEA на мові програмування Java з використанням потрібних бібліотек. Цей комплекс програмного забезпечення надає широкий спектр можливостей для розробки..

Кінцева схема роботи, яка була розроблена зображена на рисунку 3.1.

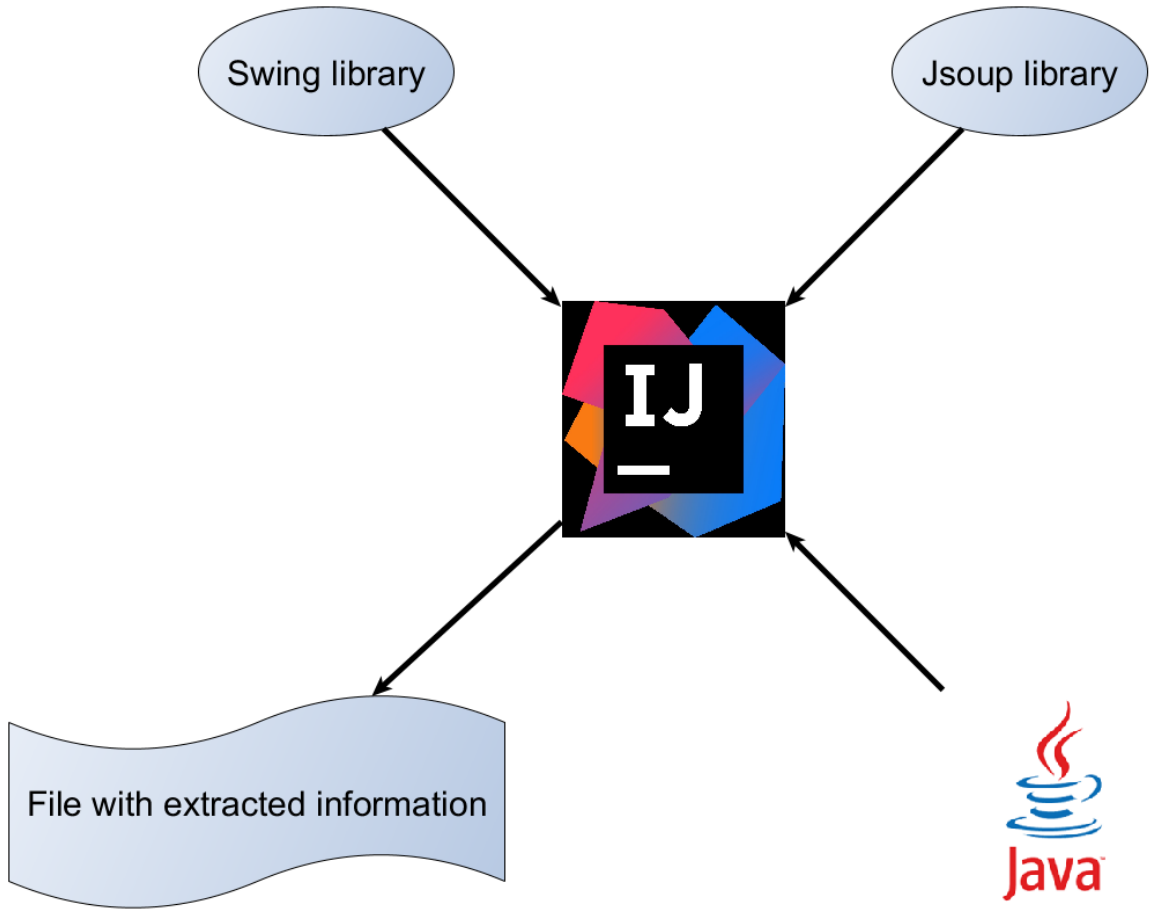


Рисунок 3.1 — Схема роботи системи

2.1 Інтегроване середовище розробки Intellij IDEA

У роботі було застосоване інтегроване середовище розробки Intellij IDEA - комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та інші) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні збірки підготовлені для Linux, Mac OS X і Windows.

2.1.1 Історія

Перша версія IntelliJ IDEA з'явилася у січні 2001 року й швидко здобула популярність, як перша Java IDE із широким набором інтегрованих інструментів для рефакторингу, що дозволяла програмістам швидко реорганізовувати сирцевий код програм. Дизайн середовища орієнтовано на продуктивність праці програмістів, дозволяючи їм сконцентруватися на розробці функціональності, тоді як IntelliJ IDEA бере на себе виконання рутинних операцій.

Починаючи з шостої версії продукту Intellij IDEA надає інтегрований інструментарій для розробки графічного користувацького інтерфейсу.

З версії 9.0 є безкоштовний вариант Community Edition з відкритими кодами. Сирцеві коди відкритої версії Intellij IDEA Community Edition поширюються в рамках ліцензії Apache 2.0. Бінарні пакунки підготовлені для Linux, Mac OS X і Windows.

До складу IntelliJ IDEA включені напрацювання, створені в результаті спільної роботи з компанією Google, яка використовувала IntelliJ IDEA як базис для своєї нового відкритого середовища розробки Android Studio. Завдяки співпраці істотно розширені штатні можливості IntelliJ IDEA з розробки застосунків для платформи Android.

2.1.2 Огляд можливостей

Community версія середовища IntelliJ IDEA підтримує інструменти (у вигляді плагінів) для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven, Ant, Gradle, мови програмування Java, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML-редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Комерційна версія «Ultimate Edition» відрізняється наявністю підтримки додаткових програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримкою технологій Java EE, UML-діаграм, підрахунок покриття коду, можливістю роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Framework і Hibernate), засобами інтеграції з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

2.2 Мова програмування Java

Java - об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток J++, що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування C# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не

пов'язана із Java.

2.2.1 Назва

Спочатку мова називалася Oak («дуб») і розроблялася Джеймсом Гослінгом для програмування побутових електронних пристроїв. Згодом вона була перейменована в Java і стала використовуватися для написання клієнтських застосунків і серверного програмного забезпечення. Названа на честь марки кави Java, яка, в свою чергу, отримала найменування однойменного острова (Ява), тому на офіційній емблемі мови зображена чашка з паркою кавою. Існує й інша версія походження назви мови, пов'язана з алюзією на каво-машину як приклад побутового устаткування, для програмування якого спочатку мова створювалася.

2.2.2 Історія

Мова програмування Java зародилася в 1991 р. в лабораторіях компанії Sun Microsystems. Розробку проекту започаткував Джеймс Ґослінґ, сам проект мав назву «Green» (Зелений). Створення першої робочої версії, яка мала назву «Oak» (дуб), зайняло 18 місяців. Оскільки виявилось, що ім'я Oak уже використовувалось іншою фірмою, то в результаті тривалих суперечок навколо назви нової мови з-поміж ряду запропонованих було вибрано назву Java, у 1995 р. мову було офіційно перейменовано.

Головним мотивом створення Java була потреба в мові програмування, яка б не залежала від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо.

Досить скоро майже всі найпопулярніші тогочасні веб-оглядачі отримали можливість запускати «безпечні» для системи Java-аплети всередині веб-сторінок. У грудні 1998 р. Sun Microsystems випустила Java 2 (спершу під назвою J2SE 1.2), де було реалізовано декілька конфігурацій для різних типів платформ. Наприклад, J2EE призначалася для створення корпоративних застосунків, а значно урізана J2ME для приладів з обмеженими ресурсами, таких як мобільні телефони. У 2006 році в маркетингових цілях версії J2 було перейменовано у Java EE, Java ME та Java SE відповідно.

13 листопада 2006 року Sun випустили більшу частину Java як вільне та відкрите програмне забезпечення згідно з умовами GNU General Public License (GPL). 8 травня 2007 корпорація закінчила процес, в результаті якого всі початкові коди Java були випущенні під GPL, за винятком невеликої частини коду, на який Sun не мала авторського права.

Період становлення Java збігся у часі з розквітом міжнародної інформаційної служби World Wide Web. Ця обставина відіграла вирішальну роль у майбутньому Java, оскільки Web теж вимагала платформо-незалежних програм.

Як наслідок, були зміщені акценти в розробці Sun з побутової електроніки на програмування для Інтернет.

2.2.3 Платформа

Під «незалежністю від архітектури» мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції.

Цього можна досягти, компілюючи початковий Java код у байт-код, який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байткод у код, пристосований до специфіки конкретної операційної системи і процесора. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами. У деяких версіях задля збільшення продуктивності JVM байт-код можна компілювати у машинний код до або під час виконання програми.

Основна перевага використання байт-коду — це портативність. Тим не менш, додаткові витрати на інтерпретацію означають, що інтерпретовані програми будуть майже завжди працювати повільніше, ніж скомпільовані у машинний код, і саме тому Java одержала репутацію «повільної» мови. Проте, цей розрив суттєво скоротився після введення декількох методів оптимізації у сучасних реалізаціях JVM.

Одним із таких методів є just-in-time компіляція (JIT), що перетворює байт-код Java у машинний під час першого запуску програми, а потім кешує його. У результаті така програма запускається і виконується швидше, ніж простий інтерпретований код, але ціною додаткових витрат на компіляцію під час виконання. Складніші віртуальні машини також використовують динамічну рекомпіляцію, яка полягає в тому, що віртуальна машина аналізує поведінку

запущеної програми й вибірково рекомпілює та оптимізує певні її частини. З використанням динамічної рекомпіляції можна досягти більшого рівня оптимізації, ніж за статичної компіляції, оскільки динамічний компілятор може робити оптимізації на базі знань про довкілля періоду виконання та про завантажені класи. До того ж він може виявляти так звані гарячі точки (англ. *hot spots*) — частини програми, найчастіше внутрішні цикли, які займають найбільше часу при виконанні. JIT-компіляція та динамічна рекомпіляція збільшує швидкість Java-програм, не втрачаючи при цьому портативності.

Існує ще одна технологія оптимізації байткоду, широко відома як статична компіляція, або компіляція *ahead-of-time* (AOT). Цей метод передбачає, як і традиційні компілятори, безпосередню компіляцію у машинний код. Це забезпечує хороші показники в порівнянні з інтерпретацією, але за рахунок втрати переносності: скомпільовану таким способом програму можна запустити тільки на одній, цільовій платформі.

Швидкість офіційної віртуальної машини Java значно покращилася з моменту випуску ранніх версій, до того ж, деякі випробування показали, що продуктивність JIT-компіляторів у порівнянні зі звичайними компіляторами у машинний код майже однакова. Проте ефективність компіляторів не завжди свідчить про швидкість виконання скомпільованого коду, тільки ретельне тестування може виявити справжню ефективність у даній системі.

2.2.4 Об’єктність

На противагу C++, Java є більш об'єктно-орієнтованою. Всі дані і дії групуються в класи об'єктів. Виключенням з повної об'єктності (як скажімо в Smalltalk) є примітивні типи (int, float тощо). Це було свідомим рішенням проектувальників мови задля збільшення швидкості. Через це Java не вважається повністю об'єктно-орієнтовною мовою.

У Java всі об'єкти є похідними від головного об'єкта (він називається просто

Object), з якого вони успадковують базову поведінку і властивості.

Хоча у C++ вперше стало доступне множинне успадкування, але у Java можливе тільки одинарне успадкування, завдяки чому виключається можливість конфліктів між членами класу (методи і змінні), які успадковуються від базових класів.

2.2.5 Безпека

У намірах проектувальників Java мала замінити C++ — об'єктного наступника мови C. Проектувальники почали з аналізу властивостей C++, які є причиною найбільшого числа помилок, щоби створити просту, безпечну і безвідмовну мову програмування.

В Java існує система винятків або ситуацій, коли програма зустрічається з неочікуваними труднощами, наприклад:

операції над елементом масиву поза його межами або над порожнім елементом

читання з недоступного каталогу або неправильної адреси URL

ввід недопустимих даних користувачем

Одна з особливостей концепції віртуальної машини полягає в тому, що помилки (виключення) не призводять до повного краху системи. Крім того, існують інструменти, які «приєднуються» до середовища періоду виконання і кожен раз, коли сталося певне виключення, записують інформацію з пам'яті для зневадження програми. Ці інструменти автоматизованої обробки виключень надають основну інформацію щодо виключень в програмах на Java.

Проте мову програмування Java не рекомендується використовувати в системах, збій в роботі яких може призвести до смерті, травм чи значних фізичних ушкоджень (наприклад, програмне забезпечення для керування атомними електростанціями, польотами, систем життєзабезпечення чи систем озброєння) через ненадійність програм, написаних на мові програмування Java (пункт ліцензії Microsoft 7.7.h.).

2.2.6 Автоматичне керування пам’яттю

Java використовує автоматичний збирач сміття (GC - Garbage Collector) для керування пам'яттю під час життєвого циклу об'єкта. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті. Проте, витік пам'яті все ж може статися, якщо код, написаний програмістом, має посилання на вже непотрібні об'єкти, наприклад на об'єкти, що зберігаються у діючих контейнерах.

Збирання сміття дозволене у будь-який час. В ідеалі воно відбувається під час бездіяльності програми. Збірка сміття автоматично форсується при нестачі вільної пам'яті в купі для розміщення нового об'єкта, що може призводити до кількасекундного зависання. Тому існують реалізації віртуальної машини Java з прибиральником сміття, спеціально створеним для програмування систем реального часу.

Java не має підтримки вказівників у стилі C/C++. Це зроблено задля безпеки й надійності, аби дозволити збирачу сміття переміщувати вказівникові об'єкти.

2.2.7 Приклад програми

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

2.3 Бібліотека Jsoup

Jsoup – бібліотека з відкритим кодом, призначена для аналізу та вилучення

та маніпулювання даними, що зберігаються в HTML документах.

HTML (англ. HyperText Markup Language — мова розмітки гіпертексту) — це мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет

Веб-браузери отримують HTML-документи з веб-сервера або з локальної пам'яті і передають документи в мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично і спочатку включені сигнали для зовнішнього вигляду документа.

Елементи HTML є будівельними блоками сторінок HTML. За допомогою конструкцій HTML, зображення та інші об'єкти, такі як інтерактивні форми, можуть бути вбудовані у візуалізовану сторінку. HTML надає засоби для створення структурованих документів, позначаючи структурну семантику тексту, наприклад заголовки, абзаци, списки, посилання, цитати та інші елементи. Елементи HTML окреслені *тегами*, написаними з використанням кутових дужок. Теги, такі як і безпосередньо вводять вміст на сторінку. Інші теги, такі як <img /> <input /> <p> оточують і надають інформацію про текст документа і можуть включати інші теги як піделементи. Браузери не показують теги HTML, але використовують їх для інтерпретації вмісту сторінки.

HTML може вбудовувати програми, написані на мові сценаріїв, наприклад JavaScript, що впливає на поведінку та вміст веб-сторінок. Включення CSS визначає вигляд і компонування вмісту. World Wide Web Consortium (W3C), які супроводжують як HTML і CSS стандартів, заохочує використання CSS над явним презентаційним HTML з 1997 року.

HTML впроваджує засоби для:

створення структурованого документа шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;

отримання інформації із Всесвітньої мережі через гіперпосилання;

створення інтерактивних форм;

включення зображень, звуку, відео, та інших об'єктів до тексту.

HTML документ. Для поліпшення взаємодії SGML вимагає аби кожна похідна мова (HTML у тому числі) визначала свою кодову таблицю для кожного документа, яка складається з *репертуару* (перелік різноманітних символів) та *позиції символу* (перелік цифрових посилань на символи з репертуару). Кожен документ HTML — це послідовність символів із репертуару.

HTML використовує найповнішу кодову таблицю UCS (англ. *Universal Character Set* — Універсальний Набір Символів).

Проте однієї кодової таблиці недостатньо для того, щоб браузери могли правильно відтворювати документи HTML. Для цього браузерам потрібно «знати» специфічну кодову таблицю документа, яку автор має зазначати завжди в елементі `meta` із параметром `charset`. За замовчуванням використовується кодова таблиця ISO-8859-1, відома також як Latin-1.

Приклад використання:

Насамперед вам необхідно отримати екземпляр класу Document з org.jsoup.nodes.Document із зазначенням на джерело для розбору. Їм може виступати як локальний файл, так і посилання.

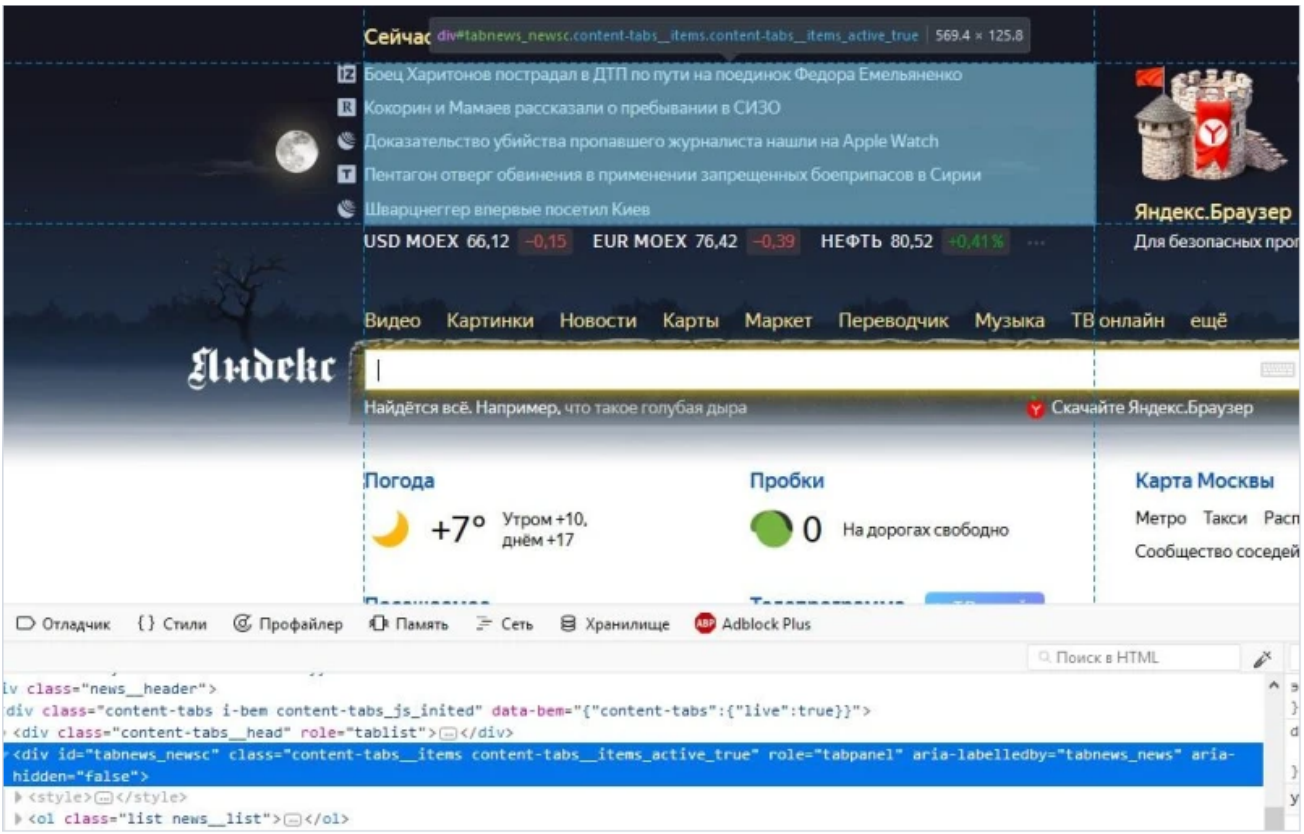
Для прикладу, ми будемо використовувати сайт yandex.ru і спробуємо отримати їх актуальну новинну стрічку:

```
Document doc = Jsoup.connect("https://yandex.ru/")
                .userAgent("Chrome/4.0.249.0 Safari/532.5")
                .referrer("http://www.google.com")
                .get();
```

User Agent є ідентифікатором, який повідомляється відвідуваного сайту. На багатьох сайтах він є найважливішим критерієм для антиспам фільтра. Referrer містить URL джерела запиту. Метод get () викликає обробляється виняток IOException, так що ми можемо обернути все в try / catch блок, або просто перекинути його далі за допомогою throws. На даний момент ми отримали вихідний код цієї сторінки. При необхідності бібліотека jsoup сама може відновити пошкоджені елементи. Тепер нам залишається лише звузити пошук до

окремого блоку.

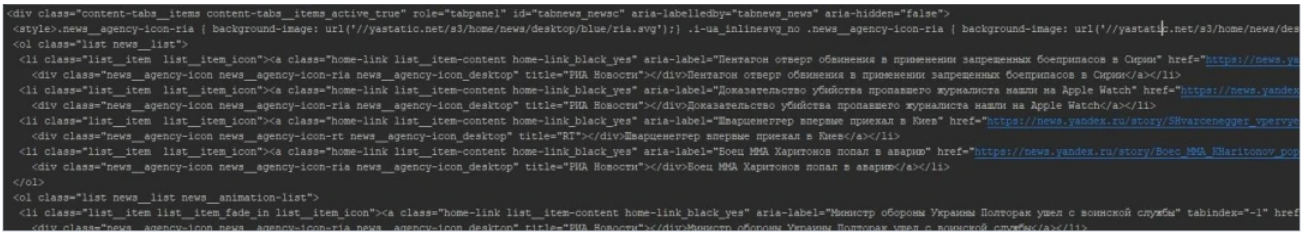
Метод select () має велику вибірку у використанні: він дозволяє шукати елементи за тегом, атрибутам, класу і іншим параметрам. Майже всі сучасні браузери підтримують можливість швидкого пошуку вихідного коду обраного елемента. Нехитрими маніпуляціями, ми знаходимо вихідний код потрібного нам елементу і отримуємо div блок із зазначеним класом, його ми і будемо використовувати для вибірки.



Скористаємося класом Elements з org.jsoup.select.Elements, для вибірки всіх елементів з нашого обраного блоку:

```
1  Elements listNews = doc.select("div#tabnews_newsc.content-tabs__items.content-tabs__items_active_true"
```

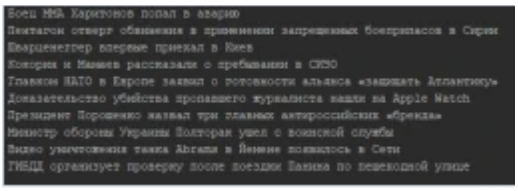
Зараз ми маємо щось на зразок цього:



Тепер нам залишається лише використовувати невеликий цикл для перебору всіх елементів:

```
for (Element element : listNews.select("a"))
            System.out.println(element.text());
```

Метод text () дозволяє відкинути код розмітки і залишає лише поєднання тексту для всіх вхідних елементів. Результат виконання буде такий:



Неважко помітити, що реальна кількість отриманих рядків не відповідає фактичному відображенню на сторінці. В цьому і полягають підводні камені.

Якщо подивитися вихідний код розмітки, можна помітити, що остання новина анімаційно змінюється з певним інтервалом часу.

Частина таких "каменів" вирішується додаткової вибіркою, ну і звичайно тестами. Може виявитися так, що перші п'ять елементів будуть містити потрібну нам інформацію, а на шостому елементі буде лише заскріптованних порожній рядок. Буває і таке, що блоки не володітимуть якимись ідентифікаторами, тоді є можливістю прямо вказати за допомогою методу get (int index) на номер позиції розглянутого елемента.

```
System.out.println(listNews.select("a").get(2).text());
```

2.4 Бібліотека Swing(Java)

Swing — інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java. Це частина бібліотеки базових класів Java (*JFC*, Java Foundation Classes).

Swing розробляли для забезпечення функціональнішого набору програмних компонентів для створення графічного інтерфейсу користувача, ніж у ранішого інструментарію AWT. Компоненти Swing підтримують специфічні *look-and-feel* модулі, що динамічно підключаються. Завдяки ним можлива емуляція графічного інтерфейсу платформи (тобто до компоненту можна динамічно підключити інші, специфічні для даної операційної системи вигляд і поведінку). Основним недоліком таких компонентів є відносно повільна робота, хоча останнім часом це не вдалося підтвердити через зростання потужності персональних комп'ютерів. Позитивна сторона — універсальність інтерфейсу створених програм на всіх платформах.

2.4.1 Історія

На початку існування Java класів Swing не було взагалі. Через слабкі місця в AWT (початковій GUI системі Java) було створено Swing. AWT визначає базовий набір елементів керування, вікон та діалогів, які підтримують придатний до використання, але обмежений у можливостях графічний інтерфейс. Однією з причин обмеженості AWT є те, що AWT перетворює свої візуальні компоненти у відповідні їм еквіваленти, що не залежать від платформи, які називаються рівноправними компонентами. Це означає, що зовнішній вигляд компонентів визначається платформою, а не закладається в Java. Оскільки компоненти AWT використовують «рідні» ресурси коду, вони називаються ваговитими (англ. *highweigh*).

Використання «рідних» рівноправних компонентів породжує деякі проблеми. По-перше, у зв'язку із різницею, що існує між операційними системами, компонент може виглядати або навіть вести себе по-різному на

різноманітних платформах. Така мінливість суперечила філософії Java: «написане один раз, працює скрізь». По-друге, зовнішній вигляд кожного компонента був фіксованим (оскільки усе залежало від платформи), і це неможливо було змінити (принаймні, це важко було зробити). По-третє, використання ваговитих компонентів тягнуло за собою появу нових обмежень. Наприклад, ваговитий компонент завжди має прямокутну форму і є непрозорим.

Незабаром після появи початкової версії Java, стало очевидним, що обмеження, властиві AWT, були настільки незручними, що потрібно було знайти кращий підхід. У результаті з'явились класи Swing як частина бібліотеки базових класів Java (JFC). В 1997 році вони були включені до Java 1.1 у вигляді окремої бібліотеки. А починаючи з версії Java 1.2, класи Swing (а також усі останні, що входили до JFC) стали повністю інтегрованимими у Java.

2.4.2 Архітектура

Незалежність від платформи: Swing — платформо-незалежна бібліотека, що означає, що програму з використанням Swing можна запустити на всіх платформах, які підтримують JVM.

Можливість для розширення: Swing — дуже розподілена архітектура, яка дозволяє «підключати» реалізації користувача вказаної інфраструктури інтерфейсів: користувачі можуть створити свою власну реалізацію цих компонентів, щоб замінити компоненти без обумовлення (*за замовчуванням*). Взагалі, користувачі Swing можуть розширити структуру, продовжуючи (з допомогою *extends*) існуючі класи і/або створюючи альтернативні реалізації основних компонентів.z

2.5 Apache POI

Проект, яким керує програмний фонд Apache, а раніше під-проект проекту Джакарта, забезпечує бібліотеки Java для читання та запису файлів у форматах Microsoft Office, таких як Word, PowerPoint та Excel.

2.5.1 Історія та дорожня карта

Назва спочатку була абревіатурою "Погана реалізація обфускування" посилаючись жартівливо на те, що формати файлів, здавалося, були навмисно затуманені, але погано, оскільки вони були успішно реінжиніровані. Це пояснення - а також схожі назви для різних під-проектів - було видалено з офіційних веб-сторінок, щоб краще продати інструменти для підприємств, які не вважали б такий гумор доцільним. Оригінальні автори (Ендрю К. Олівер та Марк Джонсон) також відзначили існування гавайської страви пої, виготовленої з пюре з кореня таро, яке мало подібні принизливі конотації.

2.5.2 Підтримка Open Office XML

POI підтримує формати файлів файлів ISO / IEC 29500: 2008 Office Open XML з версії 3.5. Вагомий внесок у підтримку OOXML прийшов від Sourcesense, компанії з відкритим кодом, яка була доручена Microsoft розробляти цей внесок. Це посилання викликало суперечки, деякі учасники POI ставлять під сумнів захист патенту POI OOXML щодо ліцензії на патент на відкриту специфікацію Microsoft.

2.5.3 Архітектура

Проект POI Apache містить такі підкомпоненти (значення абревіатур взято зі старої документації):

- POIFS (Погана файлова система впровадження обфускації) - Цей компонент зчитує та записує у форматі документу Microsoft OLE 2 Compound. Оскільки всі файли Microsoft Office - це файли OLE 2, цей компонент є основним складовим елементом для всіх інших елементів POI. Таким чином, POIFS можна використовувати для читання більш широкого спектру файлів, крім тих, чиї явні декодери вже записані в POI.
- HSSF (Horrible SpreadSheet Format) - читає і записує файли формату Microsoft Excel (XLS). Він може читати файли, написані Excel 97 і далі; цей формат файлу відомий як формат BIFF 8. Оскільки формат файлу Excel складний і містить низку складних характеристик, деякі більш досконалі функції неможливо прочитати.
- XSSF (XML SpreadSheet Format) - читає і записує файли формату Office Open XML (XLSX). Аналогічна функція встановлена для HSSF, але для файлів Office Open XML.
- HPSF (Формат набору жахливих властивостей) - читає інформацію "Зведення документа" з файлів Microsoft Office. Це, по суті, інформація, яку можна побачити, використовуючи пункт меню Файл | Властивості в додатку Office.

- HWPF (Формат жахливих текстових процесорів) - спрямований на читання та запис файлів формату Microsoft Word 97 (DOC). Цей компонент знаходиться на початкових стадіях розвитку.
- XWPF (Формат текстового процесора XML) - аналогічна функція, встановлена для HWPF, але для файлів Office Open XML.
- HSLF (Horrible Slide Format Format - чиста реалізація Java для файлів Microsoft PowerPoint. Це забезпечує можливість читання, створення та редагування презентацій (хоча деякі речі простіше зробити, ніж інші)
- HDGF (Horrible DiaGram Format) - початкова чиста реалізація Java для бінарних файлів Microsoft Visio. Він надає можливість читати вміст файлів низького рівня.
- HPBF (жахливий формат PuBlisher) - чиста реалізація Java для файлів Microsoft Publisher.
- HSMF (жахливий формат Stupid Mail) - чиста реалізація Java для файлів MSG Microsoft Outlook.
- DDF (Dreadful Drawing Format) - пакет для декодування формату малюнка Microsoft Office Drawing.

Компонент HSSF - це найдосконаліша особливість бібліотеки. Інші компоненти (HPSF, HWPF та HSLF) є зручними, але менш повнофункціональними.

Бібліотека POI також надається як розширення Ruby або ColdFusion.

Існують модулі для великих платформ даних (наприклад, Apache Hive / Apache Flink / Apache Spark), які забезпечують певну функціональність POI Apache, наприклад обробку файлів Excel.

2.6 Maven

Apache Maven — фреймворк для автоматизації збирання проектів на основі опису їх структури в файлах на мові POM (англ. Project Object Model), що є підмножиною XML. Проект Maven видається співтовариством Apache Software Foundation, де формально є частиною Jakarta Project.

Назва системи є словом з мови ідиш, сенс якої можна приблизно висловити як «збирач знання».

Maven забезпечує декларативну, а не імперативну (на відміну від засобу автоматизації збирання Apache Ant) збірку проекту. У файлах опису проекту міститься його специфікація, а не окремі команди виконання. Всі завдання по обробці файлів, описані в специфікації, Maven виконує за допомогою їх обробки послідовністю вбудованих і зовнішніх плагінів.

Maven використовується для побудови і управління проектами, написаними на Java, C #, Ruby, Scala, та іншими мовами.

Серед примітних альтернатив - система автоматичного складання Gradle, побудована на принципах Apache Ant і Maven, але використовує спеціалізований DSL на Groovy замість POM-конфігурації.

2.6.1 Історія розробки

Maven був створений канадцем Ясоном ван ЗІЛом (Jason van Zyl) і організованою їм фірмою Sonatype. Він починався як підпроект Apache Turbine в 2002 році, в 2003 році Maven був кваліфікований як Apache-проект верхнього рівня, тоді ж з'явилася його перша версія - Maven 1.x, опублікованому 13 липня 2004 року працював версія 1.0. Це відбувалося, проте, так швидко, що деякі

зокрема виявилися непродуманими, наприклад, занадто багато конфігурації, проблеми з продуктивністю. Тому концепція була доопрацьована і з 2005 року розпочалася паралельна розробка Maven 2.x, яка була здана в версії 2.0 19 жовтня 2005 року. [7] Maven 1.x не розробляється далі і обмежується підтримкою користувачів і усуненням помилок.

Розробка Maven 3.0 почалася в 2008 році. Після восьми альфа-релізів, перша бета-версія Maven 3.0 була опублікована в жовтні 2010 року. Особливу увагу було приділено її зворотної сумісності з Maven 2. Для більшості проектів перехід від версії Maven 2 до версії Maven 3 не вимагає ніяких змін.

Розробка Maven відбувається в наступних підпроектах:

- Maven 1.x і Maven 2.x - старі версії Maven.
- Maven 3.x розвиває поточну лінію продуктів Maven.
- Plugins розробляє більшість maven-плагінів.
- Shared Components виготовляє компоненти програмного забезпечення, які можуть використовуватися всіма іншими підпроекту.
- Ant Tasks дозволяє використовувати можливості Ant з Maven.
- Doxia - фреймворк для генерації контенту з форматів Almost Plain Text (APT), Confluence, DocBook, FML (FAQ Markup Language), LaTeX, Rich Text Format (RTF), TWiki, XDoc і XHTML.
- SCM (Source Code Management) розробляє програмне забезпечення для підключення Apache до різних систем версіонірованія як CVS або Subversion.
- Surefire розробляє тест-фреймворк для Maven-а.

- Wagon готує абстракцію комунікаційних протоколів як «доступ до файлів», HTTP або FTP.

2.6.2 Об'єктна модель опису проекту

Інформація для складання проекту, підтримуваного Apache Maven, міститься в XML-файлі з назвою pom.xml. При запуску Maven перевіряє, чи містить конфігураційний файл всі необхідні дані і всі дані синтаксично правильно записані.

приклад файлу pom.xml:

```
<project>
  <!-- версия модели для POM-ов Maven 2.x всегда 4.0.0 -->
  <modelVersion>4.0.0</modelVersion>

  <!-- координаты проекта, то есть набор значений, который
       позволяет однозначно идентифицировать этот проект -->

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>

  <!-- зависимости от библиотек -->

  <dependencies>
    <dependency>

      <!-- координаты необходимой библиотеки -->

      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>

      <!-- эта библиотека используется только для запуска и компилирования тестов -->

      <scope>test</scope>

    </dependency>
  </dependencies>
</project>
```

pom.xml з дипломного проекту:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>kpi.ua</groupId>
  <artifactId>Parser</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Parser</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>
UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>

     <!-- StringUtils -->
    <dependency>
          <groupId>org.apache.commons</groupId>
     <artifactId>commons-lang3</artifactId>
     <version>3.4</version>
     </dependency>

     <!-- JSOUP -->
     <dependency>
     <groupId>org.jsoup</groupId>
     <artifactId>jsoup</artifactId>
     <version>1.10.3</version>
     </dependency>

     <!-- POI -->
     <dependency>
     <groupId>org.apache.poi</groupId>
     <artifactId>poi</artifactId>
     <version>3.17</version>
     </dependency>

  </dependencies>
</project>
```

Мінімальна конфігурація включає версію конфігураційного файлу, ім'я проекту, його автора і версію. За допомогою pom.xml конфигурируются залежності від інших проектів, індивідуальні фази процесу побудови проекту (build process), список плагінів, що реалізують порядок складання. Великі проекти можуть бути поділені на кілька модулів, або підпроектів, кожен зі своїм власним POM. Операції над модулями можуть виконуватися через загальний кореневої POM єдиною командою.

POM-файли підпроектів можуть успадковувати конфігурацію від інших файлів конфігурації. У той же час всі файли конфігурації обов'язково успадковуються від «Супер POM» файлу за замовчуванням. Супер POM забезпечує конфігурацію за замовчуванням, наприклад, стандартна структура каталогів, що використовуються за замовчуванням плагіни, прив'язка до фаз життєвого циклу та інше.

2.6.3 Життєвий цикл

Життєвий цикл maven-проекту - це список названих фаз, що визначає порядок дій при його побудові. Життєвий цикл Maven містить три незалежних порядку виконання:

- clean - життєвий цикл для очищення проекту. Містить наступні фази:
  1.pre-clean
  2.clean
  3.post-clean
- default - основний життєвий цикл, що містить наступні фази:

1.validate - виконується перевірка, чи є структура проекту повну й правильну.

2.generate-sources

3.process-sources

4.generate-resources

5.process-resources

6.compile - компілюються вихідні тексти.

7.process-test-sources

8.process-test-resources

9.test-compile

10.test - зібраний код тестується заздалегідь підготовленим набором тестів.

11. package - упаковка відкомпільованих класів та інших ресурсів. Наприклад, в JAR-файл.

12.integration-test - програмне забезпечення в цілому або його великі модулі піддаються інтеграційного тестування. Перевіряється взаємодія між складовими частинами програмного продукту.

13.install - установка програмного забезпечення в локальний Maven-репозиторій, щоб зробити його доступним для інших проектів поточного користувача.

14.deploy - стабільна версія програмного забезпечення поширюється на віддалений Maven-репозиторій, щоб зробити його доступним для інших користувачів.

- site - життєвий цикл генерації проектної документації. Складається з фаз:

  1.pre-site

  2.site

  3.post-site

  4.site-deploy

2.6.4 Залежності

У файлі pom.xml задаються залежності, які мають керований за допомогою Maven проект. Менеджер залежностей заснований на декількох основних принципах:

- Репозиторії. Maven шукає необхідні файли в локальних каталогах або в локальному maven-репозиторії. Якщо залежність не може бути локально

дозволена, Maven підключається до зазначеного maven-сховища в мережі і копіює в локальний репозиторій. За замовчуванням Maven використовує Maven Central Repository [22], але розробник може конфігурувати і інші публічні Maven-репозиторії, такі, як Apache, Ibiblio, Codehaus або Java.Net.

- Транзитивні залежності. Необхідні бібліотеки завантажуються в проект автоматично. При вирішенні конфлікту версій використовується принцип «найближчій» залежності, тобто вибирається залежність, шлях до якої через список залежних проектів є найбільш коротким.
- Виключення залежностей. Файл опису проекту передбачає можливість виключити залежність у разі виявлення циклічності або відсутності необхідності в певній бібліотеці.
- Пошук залежностей. Пошук залежностей (open-source-бібліотек і модулів) ведеться за їх координатами (groupId, artifactId і version). Ці координати можуть бути визначені за допомогою спеціальних пошукових машин, наприклад, Maven search engine. Наприклад, за пошуковим ознакою «pop3» пошукова машина надає результат з groupId = "com.sun.mail" і artifactId = "pop3".
- Менеджери репозиторіїв. Репозиторії реалізуються за допомогою менеджерів репозитаріїв Maven (Maven Repository Manager), таких як Apache Archiva, Nexus (раніше Proximity), Artifactory, Codehaus Maven Proxy або Dead Simple Maven Proxy.

Область поширення залежності дозволяє включати залежності тільки на певну стадію побудови проекту. Існує 6 можливих областей:

1.compile. Область за замовчуванням. Залежність доступна на всіх дорогах пошуку класів в проекті. Поширюється на залежні проекти.

2.provided. Область аналогічна compile, за винятком того, що JDK або контейнер сам надасть залежність під час виконання програми.

3.runtime. Залежність не потрібна для компіляції, але потрібна для

виконання.

4.test. Залежність не потрібна для нормальної роботи програми, а потрібна тільки для компіляції і запуску тестів.

5.system. Область аналогічна provided за винятком того, що містить залежність JAR вказується явно. Артефакт не шукає в репозиторії.

6.import (починаючи з версії Maven 2.0.9) використовується тільки з залежністю типу pom в секції <dependencyManagement>. Залежно поточного POM замінюються на залежності із зазначеного POM.

2.7 Git

Git – розподілена система керування версіями. Проект був створений Лінус Торвальдс для управління розробкою ядра Linux, перша версія випущена 7 квітня 2005 року. На сьогоднішній день його підтримує Джун Хама.

Серед проектів, які використовують Git - ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивів Linux.

Програма є вільною і випущена під ліцензією GNU GPL версії 2. Стандартною TCP порт 9418.

2.7.1 Історія

Розробка ядра Linux велася на проприетарной системі BitKeeper, яку автор, - Ларрі Маквей, сам розробник Linux, - надав проекту з безкоштовної ліцензії. Розробники, висококласні програмісти, написали кілька утиліт, і для однієї Ендрю Тріджелл справив реверс-інжиніринг формату передачі даних BitKeeper. У відповідь Маквей звинуватив розробників в порушенні угоди і відкликав ліцензію, і Торвальдс взявся за нову систему: жодна з відкритих систем не дозволяла тисячам програмістів кооперувати свої зусилля (той же конфлікт призвів до написання Mercurial).

Ідеологія була проста: взяти підхід CVS і перевернути з ніг на голову , і заодно додати надійності.

Початкова розробка велася менше, ніж тиждень 3 квітня 2005 року розробка

почалася, і вже 7 квітня код Git керувався неготовою системою. 16 червня Linux був переведений на Git, а 25 липня Торвальдс відмовився від обов'язків провідного розробника.

2.7.2 Історія версій

Версия	Первоначальная дата выпуска	Последняя версия	Дата выпуска
0.99	2005-07-11	0.99.9n	2005-12-15
1.0	2005-12-21	1.0.13	2006-01-27
1.1	2006-01-08	1.1.6	2006-01-30
1.2	2006-02-12	1.2.6	2006-04-08
1.3	2006-04-18	1.3.3	2006-05-16
1.4	2006-06-10	1.4.4.5	2008-07-16
1.5	2007-02-14	1.5.6.6	2008-12-17
1.6	2008-08-17	1.6.6.3	2010-12-15
1.7	2010-02-13	1.7.12.4	2012-10-17
1.8	2012-10-21	1.8.5.6	2014-12-17
1.9	2014-02-14	1.9.5	2014-12-17
2.0	2014-05-28	2.0.5	2014-12-17
2.1	2014-08-16	2.1.4	2014-12-17
2.2	2014-11-26	2.2.3	2015-09-04
2.3	2015-02-05	2.3.10	2015-09-29
2.4	2015-04-30	2.4.12	2017-05-05
2.5	2015-07-27	2.5.6	2017-05-05
2.6	2015-09-28	2.6.7	2017-05-05
2.7	2015-10-04	2.7.6	2017-08-10
2.8	2016-03-28	2.8.6	2017-08-10
2.9	2016-06-13	2.9.5	2017-08-10
2.10	2016-09-02	2.10.5	2017-09-26
2.11	2016-11-29	2.11.4	2017-09-26
2.12	2017-02-24	2.12.5	2017-09-26
2.13	2017-05-10	2.13.7	2018-05-29
2.14	2017-08-04	2.14.4	2018-05-29
2.15	2017-10-30	2.15.2	2018-05-29
2.16	2018-01-17	2.16.4	2018-05-29
2.17	2018-04-03	2.17.1	2018-05-29
2.18	2018-06-21	2.18.1	2018-09-27
2.19	2018-09-10	2.19.2	2018-11-21
2.20	2018-12-09	2.20.2	2019-12-07
2.21	2019-02-24	2.21.1	2019-12-07
2.22	2019-06-07	2.22.2	2019-12-07
2.23	2019-08-16	2.23.1	2019-12-07
2.24	2019-11-04	2.24.1	2019-12-07
2.25	2020-01-13	2.25.0	2020-01-13
2.26	2020-03-23	2.26.2	2020-04-20
Легенда: Старая версия, не поддерживается	Старая поддерживаемая версия	**Текущая версия**	Тестовая версия		

2.7.3 Можливості

Система спроектована як набір програм, спеціально розроблених з урахуванням їх використання в сценаріях. Це дозволяє зручно створювати спеціалізовані системи контролю версій на базі Git або призначені для користувача інтерфейси. Наприклад, Cogito є саме таким прикладом оболонки до репозиторіїв Git, а StGit використовує Git для управління колекцією виправлень (патчів).

Git підтримує швидке поділ і злиття версій, включає інструменти для візуалізації та навігації по нелінійної історії розробки. Як і Darcs, BitKeeper, Mercurial, Bazaar і Monotone, Git надає кожному розробнику локальну копію всієї історії розробки, зміни копіюються з одного сховища в інший. Віддалений доступ до репозиторіїв Git забезпечується git-демоном, SSH- або HTTP-сервером. TCP-сервіс git-daemon входить в дистрибутив Git і є поряд з SSH найбільш поширеним і надійним методом доступу. Метод доступу по HTTP, незважаючи на ряд обмежень, дуже популярний в контрольованих мережах, тому що дозволяє використовувати існуючі конфігурації мережевих фільтрів.

2.7.4 Особливості реалізації

Ядро Git є набором утиліт командного рядка з параметрами. Всі налаштування зберігаються в текстових файлах конфігурації. Така реалізація робить Git легко портіруемость на будь-яку платформу і дає можливість легко інтегрувати Git в інші системи (зокрема, створювати графічні git-клієнти з будь-яким бажаним інтерфейсом).

Репозиторій Git є каталог файлової системи, в якому знаходяться файли конфігурації сховища, файли журналів, що зберігають операції, що виконуються над репозиторієм, індекс, що описує розташування файлів, і сховище, що містить власне файли. Структура сховища файлів не відображає реальну структуру зберігається в репозиторії файлового дерева, вона орієнтована на підвищення швидкості виконання операцій з репозиторієм.

Коли ядро обробляє команду зміни (неважливо, при локальних змінах або при отриманні патча від іншого вузла), воно створює в сховище нові файли, що відповідають новим станам змінених файлів. Істотно, що ніякі операції не змінюють вмісту вже існуючих в сховище файлів.

За замовчуванням репозиторій зберігається в підкаталозі з назвою «.git» в кореневому каталозі робочої копії дерева файлів, що зберігається в репозиторії. Будь-яке файлове дерево в системі можна перетворити в репозиторій git, віддавши команду створення сховища з кореневого каталогу цього дерева (або вказавши кореневої каталог в параметрах програми). Репозиторій може бути імпортований з іншого вузла, доступного по мережі. При імпорті нового сховища автоматично створюється робоча копія, відповідна останньому зафіксованому станом імпортованого сховища (тобто не копіюються зміни в робочій копії вихідного вузла, для яких на те вузлі не було виконане команда commit).

2.7.5 Архітектура

Нижній рівень git є так званої контентно-адресується файлової системою. Інструмент командного рядка git містить ряд команд по безпосередньої маніпуляції цим репозиторієм на низькому рівні. Ці команди не потрібні при нормальній роботі з git як з системою контролю версій, але потрібні для реалізації складних операцій (ремонт пошкодженого сховища та так далі), а також дають можливість створити на базі сховища git свій додаток.

Для кожного об'єкта в репозиторії обчислюється SHA-1-хеш, і саме він стає ім'ям файлу, що містить даний об'єкт в каталозі .git / objects. Для оптимізації роботи з файловими системами, які не використовують дерева для каталогів, перший байт хешу стає ім'ям підкаталогу, а решта - ім'ям файлу в ньому, що знижує кількість файлів в одному каталозі (обмежуючий фактор продуктивності на таких застарілих файлових системах).

Всі номери об'єкти сховища, включаючи посилання на один об'єкт, що знаходиться всередині іншого об'єкта, є SHA-1-хешамі. Крім того, в репозиторії

існує каталог refs, який дозволяє задати читаються людиною імена для якихось об'єктів Git. У командах Git обидва види посилань - читаються людиною з refs, і нижележащие SHA-1 - повністю взаємозамінні. У класичному звичайному сценарії в репозиторії git є три типи об'єктів - файл, дерево і «Комміт» (англ. Commit - фіксація). Файл є якась версія якогось призначеного для користувача файлу, дерево - сукупність файлів з різних підкаталогів, «Комміт» - дерево і якась додаткова інформація (наприклад, батьківські коммітов, а також коментар).

У репозиторії іноді проводиться прибирання сміття, під час якої застарілі файли замінюються на «дельти» між ними і актуальними файлами (тобто, актуальна версія файлу зберігається неінкрементально, інкремент використовуються тільки для повернення до попередніх версій), після чого дані «дельти» складаються в один великий файл, до якого будується індекс. Це знижує вимоги по ємності збереження. Репозиторій Git буває локальний і віддалений. Локальний репозиторій - це підкаталог .git, створюється (в порожньому вигляді) командою git init і (в непорожньої вигляді з негайним копіюванням вмісту батьківського віддаленого сховища та проставлянням посилання на батька) командою git clone.

Практично всі звичайні операції з системою контролю версій, такі, як Комміт і злиття, проводяться тільки з локальним репозиторієм. Віддалений репозиторій можна тільки синхронізувати з локальним як «вгору» (push), так і «вниз» (pull). Наявність повністю всього сховища проекту локально для кожного розробника дає Git ряд переваг перед SVN. Так, наприклад, всі операції, крім push і pull, можна здійснювати без наявності інтернет-з'єднання.

Дуже потужною можливістю git є гілки, реалізовані куди більш повно, ніж в SVN: по суті, гілка git є не більше ніж іменована посилання, яка вказує на якийсь Комміт в репозиторії (використовується підкаталог refs). Комміт без створення нової гілки всього лише пересуває це посилання на себе, а Комміт зі створенням гілки - залишає стару посилання на місці, але створює нову на новий Комміт, і оголошує її поточної. Замінити локальні девелоперські файли на набір файлів з

іншої гілки, тим самим перейшовши до роботи з нею - так само тривіально.

Також підтримуються субрепозіторіі з синхронізацією поточних гілок в них.

Команда push передає всі нові дані (ті, яких ще немає в віддаленому репозиторії) з локального сховища в репозиторій віддалений. Для виконання цієї команди необхідно, щоб віддалений репозиторій не мав нових коммітов в себе від інших клієнтів, інакше push завершується помилкою, і доведеться робити pull і злиття. Команда pull - протилежна команді push. У разі, якщо одна і та ж гілка має незалежну історію в локальній і у віддаленій копії, pull негайно переходить до злиття.

Злиття в межах різних файлів здійснюється автоматично (все це поведінка налаштовується), а в межах одного файлу - стандартним трёхпанельним порівнянням файлів. Після злиття потрібно оголосити конфлікти як дозволені.

Результатом всього цього є новий стан в локальних файлах у того розробника, що здійснив злиття. Йому потрібно негайно зробити Комміт, при цьому в даному об'єкті коммітов в репозиторії виявиться інформація про те, що Комміт є результат злиття двох гілок і має два батьківських коммітов.

Крім злиття, Git підтримує ще операцію переміщення (англ. Rebase). Ця операція є отримання набору всіх змін в галузі A, з подальшим їх «накатом» на гілку B. В результаті гілка B просувається до стану AB. На відміну від злиття, в історії гілки AB не залишиться ніяких проміжних коммітов гілки A (тільки історія гілки B і запис про самому rebase, це спрощує інтеграцію великих і дуже великих проектів).

Також Git має тимчасовий локальний індекс файлів. Це - проміжне сховище між власне файлами і черговим коммітов (Комміт робиться тільки з цього індексу). За допомогою цього індексу здійснюється додавання нових файлів (git add додає їх в індекс, вони потраплять в наступний Комміт), а також Комміт не всіх змінених файлів (Комміт робиться тільки тих файлів, яким був зроблений git add). Після git add можна редагувати файл далі, вийдуть три копії одного і того ж файлу - остання, в індексі (та, що була на момент git add), і в останньому Ком.

Ім'я гілки за замовчуванням: master. Ім'я віддаленого сховища за замовчуванням, що створюється git clone під час типової операції «взяти наявний проект з сервера собі на машину»: origin.

Таким чином, в локальному репозиторії завжди є гілка master, яка є останній локальний Комміт, і гілка origin / master, яка є останнім стан віддаленого сховища на момент завершення виконання останньої команди pull або push. Команда fetch (частковий pull) - бере з віддаленого сервера всі зміни в origin / master, і переписує їх в локальний репозиторій, просуваючи мітку origin / master.

Якщо після цього master і origin / master розійшлися в сторони, то необхідно зробити злиття, встановивши master на результат злиття (команда pull є fetch + merge). Далі можливо зробити push, відправивши результат злиття на сервер і встановивши на нього origin / master.

2.7.6 Деталі реалізації в Windows

У Windows-версії (офіційна Windows-версія називається mSysGit) використовується пакет mSys - порт POSIX-сумісної командного рядка під Windows з проекту MinGW. Під mSys перенесені всі необхідні для Git бібліотеки та інструменти, а також сам Git. При роботі з віддаленими репозиторіями по протоколу SSL використовується сховище сертифікатів з mSys, а не з Windows.

Існує чимало графічних оболонок для Git для Windows, наприклад, TortoiseGit. Всі вони реалізовані через виклики mSysGit і вимагають його установки на машину. Не виняток і SourceTree, рішення компанії Atlassian, але mSysGit воно містить в собі, що має свої плюси і мінуси (так установка в глибокий підкаталог ускладнює додавання в mSys потрібних SSL-сертифікатів).

Оскільки в Windows використовується відмінний від більшості Unix-подібних систем символ кінця рядка, для роботи колективів, які використовують різні операційні системи, передбачаються параметри (як для клієнтів, так і рівня сховища), що забезпечують уніфіковане представлення кінця рядка.

2.7.7 Мережеві можливості та серверні рішення

Git використовує мережу тільки для операцій обміну з віддаленими репозиторіями. Можливе використання наступних протоколів:

- git-протокол (схема URI - git :) - відкритий протокол, що вимагає наявності на сервері запущеного git-демона (поставляється разом з Git), протокол не має коштів аутентифікації користувачів;
- SSH (ssh :) - використовує аутентифікацію користувачів за допомогою пар ключів, а також вбудований в Unix-систему «основний» SSH-сервер (sshd), з боку сервера потрібне створення облікових записів з git в якості оболонки;
- HTTP і HTTPS (http :, https :) - використовує інструмент curl (для Windows - поставляється разом з git), і його можливості HTTP-аутентифікації, як і його підтримку SSL і сертифікатів.

В останньому випадку потрібна робота на серверній стороні веб-додатки, виконуючого роль прошарку між командами Git на сервері і HTTP-сервером (серед таких WebGitNet, розроблений на ASP.NET MVC 4). Крім підтримки серверної сторони команд push і pull, такі веб-додатки можуть також давати доступ тільки на читання до сховища через веб-браузер.

2.7.8 Графічні інтерфейси

Розроблено безліч графічних інтерфейсів для системи, серед них - GitKraken (багатоплатформовий умовно безкоштовний клієнт Git), SmartGit (багатоплатформовий інтерфейс на Java), gitk (проста і швидка програма, написана на Tcl / Tk, яка поширюється з самим Git), Giggle (варіант на Gtk +), TortoiseGit (інтерфейс, реалізований як розширення для провідника Windows), SourceTree (безкоштовний Git-клієнт для Windows і Mac), Github-клієнт і ряд інших.

Крім того, розроблено безліч веб-фронтенда, в числі яких GitWebAdmin, GitLab, Gitblit, Gerrit, Gitweb, Kallithea, Gitea.

2.8 StringUtils

Клас StringUtils надає методи для null-безпечний операцій з рядками. Багато методів цього класу мають відповідні методи, визначені в класі java.lang.String, які не є null-безпечний, Однак замість цього в цьому розділі основна увага буде приділена декільком методам, які не мають еквівалентів в класі String.

В даному класі є такі методы:

- Метод ContainsAny

Метод containsAny перевіряє, чи містить даний String який-небудь символ в даному наборі символів. Цей набір символів може бути переданий у вигляді String або char varargs.

Наступний фрагмент коду демонструє використання двох перевантажених різновидів цього методу з перевіркою результату:

```
String string = "baeldung.com";
boolean contained1 = StringUtils.containsAny(string, 'a', 'b', 'c');
boolean contained2 = StringUtils.containsAny(string, 'x', 'y', 'z');
boolean contained3 = StringUtils.containsAny(string, "abc");
boolean contained4 = StringUtils.containsAny(string, "xyz");

assertTrue(contained1);
assertFalse(contained2);
assertTrue(contained3);
assertFalse(contained4);
```

- Метод ContainsIgnoreCase

Метод containsIgnoreCase перевіряє, чи містить даний рядок String інший рядок без урахування регістру

Наступний фрагмент коду перевіряє, що String «baeldung.com» містить «BAELDUNG», коли великі та малі літери ігноруються:

```
String string = "baeldung.com";
boolean contained = StringUtils.containsIgnoreCase(string, "BAELDUNG");

assertTrue(contained);
```

- Метод CountMatches

Метод counterMatches підраховує, скільки разів символ або подстрока з'являється в даній String. Нижче приведена демонстрація цього методу, що підтверджує, що 'w' з'являється чотири рази, а «com» двічі в String «welcome to www.baeldung.com»:

```
String string = "welcome to www.baeldung.com";
int charNum = StringUtils.countMatches(string, 'w');
int stringNum = StringUtils.countMatches(string, "com");

assertEquals(4, charNum);
assertEquals(2, stringNum);
```

- Додавання і попередній метод

Методи appendIfMissing і appendIfMissingIgnoreCase додають суфікс до кінця заданого рядка String, якщо він ще не закінчується жодним з переданих суфіксів з урахуванням регістра і без урахування регістра відповідно. Точно так же методи prependIfMissing і prependIfMissingIgnoreCase додають префікс до початку заданого рядка String, якщо він не починається ні з одного з переданих префіксів. У наступному прикладі методи appendIfMissing і prependIfMissing використовуються для додавання суфікса і префікса до String «baeldung.com» без повторення цих афіксів:

```
String string = "baeldung.com";
String stringWithSuffix = StringUtils.appendIfMissing(string, ".com");
String stringWithPrefix = StringUtils.prependIfMissing(string, "www.");

assertEquals("baeldung.com", stringWithSuffix);
assertEquals("www.baeldung.com", stringWithPrefix);
```

- Метод зміни регістру

Клас String вже визначає методи для перетворення всіх символів String в верхній або нижній регістр. Цей підрозділ тільки ілюструє використання методів, які змінюють регістр String іншими способами, включаючи swapCase, capitalize і uncapitalize.

Метод swapCase змінює регістр String, змінюючи великі букви в нижній

регістр і нижній регістр у верхній регістр:

```
String originalString = "baeldung.COM";
String swappedString = StringUtils.swapCase(originalString);

assertEquals("BAELDUNG.com", swappedString);
```

Метод capitalize перетворює перший символ заданого String в верхній регістр, залишаючи все решта символи незмінними:

```
String originalString = "baeldung";
String capitalizedString = StringUtils.capitalize(originalString);

assertEquals("Baeldung", capitalizedString);
```

Метод uncapitalize перетворює перший символ заданого String в нижній регістр, залишаючи все решта символи незмінними:

```
String originalString = "Baeldung";
String uncapitalizedString = StringUtils.uncapitalize(originalString);

assertEquals("baeldung", uncapitalizedString);
```

- Метод звернення

Клас StringUtils визначає два методи для звернення рядків: reverse і reverseDelimited. Метод reverse переупорядочівать все символи String в зворотному порядку, а метод reverseDelimited переупорядочівать групи символів, розділених зазначеним роздільником. Наступний фрагмент коду перевертає рядок «baeldung» і перевіряє результат:

```
String originalString = "baeldung";
String reversedString = StringUtils.reverse(originalString);

assertEquals("gnudleab", reversedString);
```

За допомогою методу reverseDelimited символи звертаються в групи, а не окремо:

```
String originalString = "www.baeldung.com";
String reversedString = StringUtils.reverseDelimited(originalString, '.');

assertEquals("com.baeldung.www", reversedString);
```

- Метод Rotate

Метод rotate () циклічно зсуває символи String на кілька позицій. Фрагмент коду

нижче переміщує всі символи __String «baeldung» на чотири позиції вправо і перевіряє результат:

```
String originalString = "baeldung";
String rotatedString = StringUtils.rotate(originalString, 4);

assertEquals("dungbael", rotatedString);
```

- Метод Difference

Метод difference порівнює два рядки, повертаючи залишок другого рядка String, починаючи з позиції, в якій вона відрізняється від першої. Наступний фрагмент коду порівнює два рядки String:

«Підручники Baeldung» і «Курси Baeldung» в обох напрямках і підтверджують результат:

```
String tutorials = "Baeldung Tutorials";
String courses = "Baeldung Courses";
String diff1 = StringUtils.difference(tutorials, courses);
String diff2 = StringUtils.difference(courses, tutorials);

assertEquals("Courses", diff1);
assertEquals("Tutorials", diff2);
```

3 ГІДРОАКУСТИКА

Гідроакустика - розділ акустики, що вивчає випромінювання, прийом і поширення звукових хвиль в реальному водному середовищі (в океанах, морях, озерах і т. д.) для цілей підводного локації, зв'язку і т. п. Головна особливість підводних звуків - їх мале загасання, внаслідок чого під водою звуки можуть поширюватися на значно більші відстані, ніж, наприклад, в повітрі. Крім загасання, обумовленого властивостями самої води, на дальність поширення звуків під водою впливають рефракція звуку, його розсіювання і поглинання різними неоднорідностями середовища, обумовлені різницею температур,

солоності або щільності води.

Важливою складовою формування в реальному часі повної картини обстановки в гавані є підводні датчики. Їх різноманіття на світовому ринку представлено стаціонарними і мобільними гидроакустическими (активними і пасивними) засобами (сонарами), а також неакустіческімі сенсорами, головним чином електричними потенційними (UEP, underwater Electric potential) і магнітними датчиками, кожен з яких має певні особливості. Для повного охоплення всіх можливих загроз і цілей в акваторії військово-морської бази необхідно розміщувати безліч стаціонарних гідроакустичних споруд. Але при цьому може виникнути проблема звукового відображення, взаємовпливу і т.п. Тому необхідно обережно ставитися до оцінки рекламних тактико-технічних характеристик окремо взятих гідроакустичних станцій при спільному використанні багатьох джерел випромінювання. Це стосується не тільки стаціонарних систем виявлення, застосування яких зазвичай передує вивчення місцевих умов і станів гідрографії, але і мобільних систем бойових кораблів, які стикаються з мінливими тактичними і екологічними сценаріями. Призначені для розміщення на борту корветів і фрегатів активні сонари середнього діапазону частот можуть забезпечити на більш значних відстанях достатню роздільну здатність по надводних цілях, субмарин і засобів висадки плавців. За допомогою таких гідроакустичних засобів військовий корабель може захиститися при базуванні в гавані, однак середня частота сигналів не є оптимальною для виявлення водолазів.

Акустичний тиск, створене передавачем потужного акустичного випромінювання з борту корабля, може стати перешкодою іншим системам підводного спостереження, хоча в той же час являє собою стримуючий засіб або навіть зброю проти будь-якого водолаза, який спробує працювати поблизу судна. Тому використання таких сонаров в акваторії бази можливо лише в обмежені періоди часу, наприклад, під час якихось спеціальних подій або при наявності достовірних відомостей про очікуване терористичний напад.

Деякі проекти передбачають також використання системи

радіогідроакустичних буїв-датчиків, здатних закрити доступ до гаваням. Вони можуть розташовуватися на грунті або підвішуватися до бакену. У разі стаціонарного або напівстаціонарні розгортання для електроживлення таких акустичних сенсорів використовуються берегові джерела електроенергії.

У морському середовищі гідроакустична інформація про стан підводної обстановки знаходиться у вигляді акустичних полів. Це можуть бути акустичні поля, фізичними характеристиками яких є акустичний тиск та коливальна швидкість. Ці поля існують у вигляді акустичних хвиль, що в залежності від умов поширення поділяють на плоскі, циліндричні та сферичні хвилі. На поширення таких хвиль в морському середовищі суттєвий вплив здійснюють гідрофізичні поля морського середовища, що характеризуються значною просторово-часовою мінливістю. У цілому ж акустичне поле в будь-якій точці морського середовища формується в результаті таких процесів поширення акустичних хвиль, як рефракція, інтерференція, дифракція, відбиття, поглинання та затухання. У кількісному відношенні наведені фізичні явища описуються за допомогою таких фізичних характеристик морського середовища, як коефіцієнти заломлення, поглинання, відбиття та розсіювання звуку морською поверхнею, водними масами та морським дном. Саме взаємодія наведених явищ між собою в морському середовищі при певному розміщенні систем «НК-ГАС» і шуканих об’єктів, що визначають стан підводної обстановки в контрольованому морському середовищі, відносно один одного та морських границь і обумовили появу акустичних особливостей цього середовища, які суттєвим чином впливають на характер розміщення гідроакустичної інформації про підводну обстановку. Розглянемо більш детально акустичні особливості морського середовища. На великих глибинах розподіл швидкості поширення звуку за глибиною є дуже різноманітним. У зв’язку з цим визначають два протилежні випадки поширення гідроакустичної інформації: антихвильовий і хвильовий. У першому випадку має місце інтенсивний відбір частини енергії в шарах, що лежать нижче. У другому випадку значна частина енергії гідроакустичного сигналу утримується каналом і поширюється на великі відстані.

Антихвильове поширення гідроакустичної інформації.	Антихвильовому поширенню гідроакустичної інформації відповідає зменшення швидкості звуку за глибиною, обумовлене зниженням температури. Частіше всього це відбувається внаслідок інтенсивного прогрівання верхніх шарів морського середовища під впливом сонячної радіації. На променевій картині всі промені повертають вниз. Можливі шляхи зміни морської області, з якої може бути отримана гідроакустична інформація, полягають в переміщенні гідроакустичної антени ГАС системи «НКГАС» із зони тіні або зменшенні робочої частоти ГАС
Підводний звуковий канал.	Підводний звуковий канал (ПЗК) характеризується регулярною зміною позитивної та від'ємної рефракції променів з повним внутрішнім відбиттям на горизонтах, де швидкість звуку більша мінімального значення на осі каналу. ПЗК має важливу акустичну особливість з погляду наявності гідроакустичної інформації, яка полягає в тому, що порівняно із сферичним законом поширення звукових хвиль рівень гідроакустичного сигналу в хвилеводі збільшується до 10 дБ на порядок відстані його поширення. ПЗК буває кількох типів.

Зараз лідером з продажу морського обладнання є Teledyne Marine. Teledyne Marine пропонує повний спектр послуг та технологій для застосувань безпеки. Наші перевірені на місцях продукти та можливості розробки протягом десятиліть задовольняють потреби урядів та силовиків у програмах, які включають:

- Підтримка та підтримка дайвінгу

- Утилізація вибухонебезпечних постанов (EOD)
- Безпека порту та гавані
- Підтримка корпусної та морської інфраструктури
- Пошук, рятування, відновлення тощо.

Технологічні рішення Teledyne включають безліч компактних підводних і наземних транспортних засобів, датчики, рішення для взаємозв'язку, а також засоби візуального та акустичного візуалізації для вимогливих програм безпеки. Перегляньте наші можливості, щоб дізнатися більше.

У межах гідрографії Teledyne Marine пропонує асортимент продукції для картографування морського дна від надзвичайно глибокої води до мілководдя. У цьому діапазоні Teledyne Marine може відповідати вимогам клієнта щодо розміру, простоти використання та продуктивності з якісним пакетом відповідно до бюджету. Також можуть бути аксесуари, що сприяють постачанню повного рішення, починаючи від датчиків швидкості звуку, кронштейнів, наборів кріплення, гондоли та кабелів до компенсації руху та систем INS, включаючи станцію обробки, встановлення та остаточну передачу, щоб кваліфікувати та забезпечити оптимальну роботу кінцевої системи. надані компанією. Програмні рішення Teledyne PDS пропонують пакети "під ключ" для однолучевих ехолосок Teledyne Marine, багатопроменевих ехолотів та багатопроменевих скануючих сонарів. Всі сонарні рішення виробляють стандартні стандартні дані для взаємодії з усіма основними пакетами збору даних гідрографічних.

У межах гідрографії Teledyne Marine пропонує асортимент продукції для картографування морського дна від надзвичайно глибокої води до мілководдя. У цьому діапазоні Teledyne Marine може відповідати вимогам клієнта щодо розміру, простоти використання та продуктивності з якісним пакетом відповідно до бюджету. Також можуть бути аксесуари, що сприяють постачанню повного рішення, починаючи від датчиків швидкості звуку, кронштейнів, наборів кріплення, гондоли та кабелів до компенсації руху та систем INS, включаючи станцію обробки, встановлення та остаточну

передачу, щоб кваліфікувати та забезпечити оптимальну роботу кінцевої системи. надані компанією. Програмні рішення Teledyne PDS пропонують пакети "під ключ" для однолучевих ехолосок Teledyne Marine, багатопроменевих ехолотів та багатопроменевих скануючих сонарів. Всі сонарні рішення виробляють стандартні стандартні дані для взаємодії з усіма основними пакетами збору даних гідрографічних.

4 ВИСНОВКИ

Відповідно до теми завдання: «Формування інформаційних портретів морських об’єктів на підставі аналізу змісту відповідних веб-сайтів» було створено парсер, який дає можливість витягувати дані про кораблі з веб-сайту та зберігати їх. Невід’ємною частиною роботи було створення інтерфейсу додатку для більш зручного користування їм. Також було ретельно досліженно веб-сайт, а саме vesselfinder.com/ru, для того щоб повторити його поведінку.

У ході виконаної роботи був отриманий досвід роботи з бібліотеками для роботи із текстом, для заповнення файлу типу Excel, для десктопного інтерфейсу та для парсингу, а саме StringUtils, Apache POI, Swing та Jsoup відповідно. Додаток успішно та коректно виконує свою роботу.

Отже, як результат маємо программу, що автоматизовано, з великою швидкістю витягує данні та записує їх у файл.

Розробка, що була виконана у ході цієї роботи може бути успішно використана та корисна для НТУУ «КПІ».

5 СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Learning Java [Електронний ресурс]. — NetBeans, 2015. — P. 3-16. — Режим доступу: https://netbeans.org/kb/articles/learn-java.html
2. Use Java 8 language features [Електронний ресурс]. — Режим доступу: https://developer.android.com/studio/write/java8-support
3. Documents open method [Електронний ресурс] – Режим доступу до ресурсу: https://msdn.microsoft.com/ruru/library/microsoft.office.interop.word.documents.open.aspx.
4. System Diagnostics [Електронний ресурс] – Режим доступу до ресурсу: https://msdn.microsoft.com/ru-ru/library/system.diagnostics(v=vs.110).aspx
5. Давыдов С.В. Intellij idea. Профессиональное программирование

на Java / С.В. Давыдов, А.А. Ефимов. — СПб.: БХВ-Петербург, 2005. — 800 с.

6. Gosling J. The Java Language Environment: Contents. A White Paper [Електронний ресурс] / James Gosling, Henry McGilton // Sun Microsystems, May 1996. — Режим доступу: https://www.oracle.com/technetwork/java/langenv-140151.htm
7. MVC: Model, View, Controller | Codecademy [Електронний ресурс] – Режим доступу: https://www.codecademy.com/articles/mvc.
8. Шаховська Н.Б. Програмне та алгоритмічне забезпечення сховищ та просторів даних : монографія [Текст] / Н.Б. Шаховська. – Львів : Видво Львівської політехніки, 2010.– 194 с.
9. What is: Apache [Електронний ресурс] — Режим доступу: https://www.wpbeginner.com/glossary/apache/
10. Рефакторинг: улучшение существующего кода. / Фаулер М. // Пер. с англ. СПб: Символ-Плюс – 2003. – 432 с.

Додаток 1

Формування інформаційних портретів морських об’єктів на підставі аналізу змісту відповідних Веб-сайтів

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ62212_20Б

Аркушів 1

Київ – 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ6 2212_20Б 81-1	Черкас_Б_Т_ТМ62.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ6 2212_20Б 12-1	Parser.java	Модуль логіки
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ6 2212_20Б 12-2	Writer.java	Модуль запису в файл
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ6 2212_20Б 12-3	StartUI.java	Модуль початкового інтерфейсу
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ6 2212_20Б 12-4	App.java	Модуль main

Додаток 2

Формування інформаційних портретів морських об’єктів на підставі аналізу змісту відповідних Веб-сайтів

Лістинг програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ62212_20Б

Аркушів 3

Київ – 2020

```
package kpi.ua.model;

import java.io.IOException;
import java.util.ArrayList;

import org.apache.commons.lang3.StringUtils;
import org.apache.commons.lang3.tuple.Pair;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

public class Parser {
	public void parse(String keyword){
		ArrayList<ArrayList<Pair<String, String>>> productsList =
getProductsList(keyword);
		Writer.writeIntoExcel(productsList, keyword);
	}

	private ArrayList<ArrayList<Pair<String, String>>>
getProductsList(String keyword){
		ArrayList<ArrayList<Pair<String, String>>> productsList = new
ArrayList<ArrayList<Pair<String, String>>>();

		int page = 1;
		int lastPageNumber = 0;

		do {
			System.out.println(page + "start");
			Document content = getResponsePage(page, keyword);

			if (content.text().isEmpty()) {
				return productsList;
			}

			if (page == 1)
			{
				lastPageNumber = getLastPageNumber(content);
			}

			productsList.addAll(parsePage(content));
			System.out.println(page + "finish");
			page++;
		} while( page <= lastPageNumber);

		return productsList;
	}

	private Document getResponsePage(int page, String keyword) {

		Document content = null;
		try {
```

```
                content = Jsoup.connect(createSearchUrl(page,
keyword)).get();
           } catch (IOException e) {
                e.printStackTrace();
           }

           return content;
     }

     private ArrayList<ArrayList<Pair<String, String>>>
parsePage(Document content){
           ArrayList<ArrayList<Pair<String, String>>> productsList = new
ArrayList<ArrayList<Pair<String, String>>>();
           Element tbody =
content.getElementsByAttributeValueMatching("class", "results table is-
hoverable is-fullwidth").first().getElementsByTag("tbody").first();
           Elements productElements = tbody.getElementsByTag("tr");
           for (Element productElement : productElements) {
                System.out.println("product");
                String productLink =
productElement.getElementsByAttributeValueMatching("class", "ship-
link").attr("href");
                if (StringUtils.isNotBlank(productLink))
                {
                     productLink = createProductUrl(productLink);
                     productsList.add(parseProduct(productLink));
                }
           }
           return productsList;
     }

     private ArrayList<Pair<String, String>> parseProduct(String
productLink) {
           Document content = null;
           try {
                content =
Jsoup.connect(createProductUrl(productLink)).get();
           } catch (IOException e) {
                e.printStackTrace();
           }

           Elements infoRows = new Elements();
           Elements tables =
content.getElementsByAttributeValueMatching("class", "tparams");
           for (Element table : tables) {
                infoRows.addAll(table.getElementsByTag("tr"));
           }

           return parseTableInfo(infoRows, productLink);
     }

     private ArrayList<Pair<String, String>> parseTableInfo(Elements
infoRows, String productLink) {
           ArrayList<Pair<String, String>> productInfo = new
ArrayList<Pair<String, String>>();
           productInfo.add(Pair.of("Link", productLink));
```

```
            for(Element row : infoRows) {
                  Elements key =
row.getElementsByAttributeValueMatching("class", "n3");
                  Elements value =
row.getElementsByAttributeValueMatching("class", "v3");
                  if (key.size() > 0 && value.size() > 0) {
                        productInfo.add(Pair.of(key.get(0).text(),
value.get(0).text()));
                  }
            }

            return productInfo;
      }

      private String createProductUrl(String link) {
            String baseUrl = "https://www.vesselfinder.com";
            return StringUtils.startsWithIgnoreCase(link, baseUrl) ? link :
baseUrl + link;
      }

      private String createSearchUrl(int page, String keyword) {
            String urlTemplate =
"https://www.vesselfinder.com/ru/vessels?page=%s&name=%s";
            return String.format(urlTemplate, page, keyword);
      }

      private int getLastPageNumber(Document content)
      {
            String text =
content.getElementsByAttributeValueMatching("class", "column vfix
pagination top").first().getElementsByTag("span").first().text();
            text = StringUtils.substringAfterLast(text, "/");
            text = StringUtils.trim(text);
            if (StringUtils.isNumeric(text))
            {
                  return Integer.parseInt(text);
            }

            return 0;
      }
}
```

Додаток 3

Формування інформаційних портретів морських об’єктів на підставі аналізу змісту відповідних Веб-сайтів

Опис програмного коду

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ62212_20Б

Аркушів 7

Київ – 2020

Анотація

Додаток надає користувачеві можливість отримати важливу інформацію з веб-сайту у лічені хвилини для подальшого її дослідження.

Розроблене програмне забезпечення витягує інформацію про кораблі та зберігає у вигляді файлу excel розширення після введення початкових даних.
Веб система була розроблена за допомогою платформи Intellij Idea з використанням мови програмування Java та бібліотек Jsoup, StringUtils, Swing, Apache POI та StringUtils.

ЗМІСТ

-3-

Загальні відомості

Відповідно до теми дипломної роботи, програма називається парсером(синтаксичним аналізатором)».
Програма працює через десктопний інтерфейс та потребує доступу до мережі інтернет, але не потребує встановлення на ПК зайвого ПО, що забезпечує зручність та швидкість.
Система була написана мовою Java з використанням бібліотек Jsoup, StringUtils, Swing, Apache POI та StringUtils.

-4-

Функціональне призначення

Розроблений програмний засіб повинен вирішити задачу швидкого зволікання, структурування та збереження даних. Це було реалізовано за допомогою кількох функцій системи, а саме:

- введення початкових даних;
- парсинг веб-сайту за допомогою розробленого алгоритму;
- структурування та збереження даних в файл типу excel.

-5-

Опис логічної структури

Загальний принцип роботи додатку такий:

1. Користувач вводить початкові дані;
2. Натискає на кнопку “Start”;
3. Метод, що викликається в обробнику починає парсинг;
4. Програма витягує дані з веб-сайту та структурує їх;
5. Дані зберігаються в файл типу excel.

В першу чергу інтерфейс вводу початкових даних, а саме ключового слова. Користувач вводить ключове слово.
Після натискання на кнопку “Start”, що розміщена снизу вікна інтерфейсу, викликається метод “parse”, в який передається ключове слово. Він виконує запити до сайту та працює с його відповідями, а саме зволікає дані та структурує іх. Далі дані зберігаються у файл типу excel. Після успішного завершення виводиться відповідне повідомлення у вигляді спливаючого вікна.
Після цього користувач може переглянуті отримані дані у вигляді таблиці. Файл зберігається у директорію, де знаходиться сам файл программи типу JAR.

-6-

Використовувані технічні засоби

Для організації доступу до програмного продукту потрібно мати комп’ютер або ноутбук.
Кінцевим користувачам для роботи з програмою потрібно мати тільки встановлену java.

-7-

Вхідні і вихідні дані

Вхідними даними є:

- ключове слово, за яким буде відбуватися пошук на веб-сайті.

Вихідними даними є:

- дані у згенерованому файлі типу excel.